Open Source Software Development*

Walt Scacchi

Institute for Software Researcher

University of California, Irvine

Irvine, CA 92697-3455 USA

Abstract

This article examines and reviews what is known so far about free/open source software development (FOSSD). FOSSD is not the same as software engineering as that is portrayed in common textbooks. Instead, it is a complementary approach to address the many challenges that arise in the development of complex software systems that are often built outside of a traditional corporate software development environment. This article highlights some of the basic understandings for how FOSSD works based on empirical studies of FOSSD projects, processes, and work practices in different communities. This includes identification of different types of informal online artifacts that facilitate and constrain FOSSD projects. This article also identifies what different studies examine as well as the approaches used to sample and systematically study FOSSD. Next, opportunities for constructive crossover studies of software engineering and FOSSD help reveal new directions for further research study. Finally, the last section presents limitations and conclusions regarding studies of FOSSD.

Keywords: open source software, free software, FLOSS, free/open source software development

* Revised version appears in *Encyclopedia of Software Engineering*, Taylor and Francis, 614-626, 2011.

1. Introduction

This articles examines practices, patterns, and processes that have been observed in studies of free/open source software development (FOSSD) projects. FOSSD is a way for building, deploying, and sustaining large software systems on a global basis, and differs in many interesting ways from the principles and practices traditionally advocated for software engineering [26]. Thousands of FLOSS systems are now in use by thousands to millions of end-users, and some of these FLOSS systems entail hundreds-of-thousands to millions of lines of source code. So what’s going on here, and how is FOSSD being used to build and sustain these projects different, and how might differences be employed to explain what's going on with FOSSD, and why.

One of the more significant features of FOSSD is the formation and enactment of complex software development processes and practices performed by loosely coordinated software developers and contributors [24]. These people may volunteer their time and skill to such effort, and may only work at their personal discretion rather than as assigned and scheduled. However, increasingly, software developers are being assigned as part of their job to develop or support FLOSS systems, and thus to become involved with FOSSD efforts. Further, FLOSS developers are generally expected (or prefer) to

Report Documentation Page	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **2011**	2. REPORT TYPE	3. DATES COVERED **00-00-2011 to 00-00-2011**
4. TITLE AND SUBTITLE **Open Source Software Development**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **University of California, Irvine,Institute for Software Research,Irvine,CA,92697-3455**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR’S ACRONYM(S)
		11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
This article examines and reviews what is known so far about free/open source software development (FOSSD). FOSSD is not the same as software engineering as that is portrayed in common textbooks. Instead, it is a complementary approach to address the many challenges that arise in the development of complex software systems that are often built outside of a traditional corporate software development environment. This article highlights some of the basic understandings for how FOSSD works based on empirical studies of FOSSD projects, processes, and work practices in different communities. This includes identification of different types of informal online artifacts that facilitate and constrain FOSSD projects. This article also identifies what different studies examine as well as the approaches used to sample and systematically study FOSSD. Next, opportunities for constructive crossover studies of software engineering and FOSSD help reveal new directions for further research study. Finally, the last section presents limitations and conclusions regarding studies of FOSSD.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**21**	

provide their own computing resources (e.g., laptop computers on the go, or desktop computers at home), and bring their own software development tools with them. Similarly, FLOSS developers work on software projects that do not typically have a corporate owner or management staff to organize, direct, monitor, and improve the software development processes being put into practice on such projects. But how are successful FOSSD projects and processes possible without regularly employed and scheduled software development staff, or without an explicit regime for software engineering project scheduling, budgeting, staffing or management [24]? What motivates software developers to participate in FOSSD projects? Why and how are large FOSSD projects sustained? How are large FOSSD projects coordinated, controlled or managed without a traditional project management team? Why and how might the answers to these questions change over time? These are the kinds of questions that will begin to be addressed in this article.

The remainder of this article is organized as follows. The next section provides a brief background on what FOSS is and how free software and open source software development efforts are similar and different. From there attention shifts to identify different types of informal online artifacts that facilitate and constrain FOSSD projects. The article also identifies what different studies examine as well as the approaches used to sample and systematically study FOSSD. Next, opportunities for constructive crossover studies of software engineering and FOSSD help reveal new directions for further research study. Finally, the article concludes with a discussion of limitations and constraints in the FOSSD studies so far regarding what is known about FOSSD through systematic empirical study.

2. What is free/open source software development?

Free (as in freedom) software and open source software are often treated as the same thing [8]. However, there are differences between them with regards to the licenses assigned to the respective software. Free software generally appears licensed with the GNU General Public License (GPL), while OSS may use either the GPL or some other license that allows for the integration of software that may not be free software. Free software is a social movement, whereas FOSSD is a software development methodology, according to free software advocates like Richard Stallman and the Free Software Foundation [10]. Yet some analysts also see OSS as a social movement distinguishable from but surrounding the free software movement [6], as in a set to sub-set relationship. The hallmark of free software and most OSS is that the source code is available for public access, open to study and modification, and available for redistribution to others with few constraints, except the right to ensure these freedoms. OSS sometimes adds or removes similar freedoms or copyright privileges depending on which OSS copyright and end-user license agreement is associated with a particular OSS code base. More simply, free software is always available as OSS, but OSS is not always free software. This is why it often is appropriate to refer to FOSS or FLOSS (L for *Libre*, where the alternative term “libre software” has popularity in some parts of the world) in order to accommodate two similar or often indistinguishable approaches to software development. Thus at times it may be appropriate to distinguish conditions or events that are generally associated or specific to either free software development or OSSD, but not both. Subsequently, for the purposes of this article, focus is directed at FOSSD practices, processes, and dynamics, rather than to software licenses though such licenses may impinge on them. However, when appropriate, particular studies examined in this review may be framed in terms specific to either free software or OSS when such differentiation is warranted.

FOSSD is mostly not about software engineering (SE), at least not as SE is portrayed in modern SE textbooks [e.g., 26]. FOSSD is also not SE done poorly. It is instead a different approach to the development of software systems where much of the development activity is openly visible, and development artifacts are publicly available over the Web. Furthermore, substantial FOSSD effort is directed at enabling and facilitating social interaction among developers (and sometimes also end-users), but generally there is no traditional software engineering project management regime, budget or schedule. FOSSD is also oriented towards the joint development of an ongoing community of developers and users concomitant with the FLOSS system of interest.

FLOSS developers are typically also end-users of the FLOSS they develop, and other end-users often participate in and contribute to FOSSD efforts. There is also widespread recognition that FOSSD projects can produce high quality and sustainable software systems that can be used by thousands to millions of end-users [18]. Thus, it is reasonable to assume that FOSSD processes are not necessarily of the same type, kind, or form found in modern SE projects [cf. 24, 26]. While such approaches might be used within an SE project, there is no basis found in the principles of SE laid out in textbooks that would suggest SE projects typically adopt or should practice FOSSD methods. Subsequently, what is known about SE processes, or modeling and simulating SE processes, may not be equally applicable to FOSSD processes without some explicit rationale or empirical justification. Thus, it is appropriate to review what is known so far about FOSSD.

2.1 The Early Days of Software Source Code Sharing

The sharing, modification, and redistribution of open software source code did not begin with the appearance of the first Linux software in the early 1990's, nor with the appearance of the Emacs text editor or Gnu C Compiler, GCC (later to evolve into the Gnu Compiler Collection, but still called GCC) by Richard Stallman in the mid 1980's. Instead, the sharing of open software source code dates back to the early days of computing in the 1950's. IBM established *SHARE* as a community of software developers and users of its early mainframe computers, who were aided in the efforts to create, share, enhance, reuse, and distribute open software source code [1]. IBM supported SHARE and open software source code until it decided to unbundle software purchases from its mainframe computer sales efforts, as being in its best interest. The early days of operating systems like *TENEX* for the DecSystem-10 mainframe and later for *Unix* on the DEC PDP-11 series mini-computers in the 1970's also benefited from access, study, modification, and sharing of modifications made by user-developers primarily in the computer science research community. Similarly, efforts to share, reuse, study, enhance, and redistribute then new programming languages interpreters or compilers for languages like *Lisp*, *Pascal*, and *C* again with the academic research community in the 1970-1980's was common, as no computer manufacturers showed interest in developing or sustaining such software until their commercially viability as products could be substantiated. Even complete relational database management systems like *Ingres* from UC Berkeley first appeared as open software source code that was available in the public domain in this same time period. Similarly, scientific computing routines were widely shared and reused, and their source code was published in books [20], as such tradition of publishing open application software source code for research or play purposes had already been established and in practice in the 1960's, such a Spencer's *Game Playing With Computers* [27], and the accessible through Internet-based online repositories with the computer-aided design package, *BRL-CAD* in 1983. Finally, large, integrated software development environments such as the *USC System Factory Project* incorporated hypertext facilities for navigating software sources, and for organizing and storing hypertext-linked software artifacts in team-based, multi-project repositories, also were developed for open access, sharing, study, enhancement, and redistribution over the Internet using minimally restrictive academic software licenses [5,21]. So the practice of sharing open source software source code that was accessible, available for study and modification, and with some but not universal accommodations for redistributing modified versions the source were all established practices by the early-mid 1980's. But the mid-1980's to the early 1990's is the period most often identified as the origins of the FOSS, the Free Software Movement, as documented in many forms, including the documentary film, *Revolution OS* [19]. Subsequently, by the mid-late 1990's we began to see the first "revolutionary" writings and studies about the practice of FOSSD that utilize the World-Wide Web, the GPL software license, and free software as a social movement [3], though nearly all of it somehow unaware, forgetful, or ignoring the open software efforts preceding it.

2.2 The Rise of Empirical Studies of FOSSD

The presence of the so-called revolution in software development that FOSS was said to represent challenged many scholars to think about what whether what was known about software development practices was still accurate, or whether something new was at hand. These concerns then gave rise to burst of new studies of FOSSD projects and practices to help determine what the revolution was all about.

Studies and recommended practices for FOSSD often focus on the interests, motivations, perceptions, and experiences of developers or end-user organizations. Typically, attention is directed to the *individual agent* (most commonly a person, unitary group, firm, or some other stakeholder, but sometimes a software system, tool, or artifact type) acting within a larger actor group or community. Individual behavior, personal choices, or insider views might best be analyzed, categorized, and explained in terms of volunteered or elicited statements of their interests, motivations, perceptions, or experiences. Popular treatments of OSS development [3] and Free software development [9,10], provide insight and guidance for how FLOSS development occurs, based on the first-hand experiences of those authors.

Other authors informed by such practitioner studies and informal industry/government polls (like those reported in *CIO Magazine,* the Open Source Business Conference, the Open Solutions Alliance, and elsewhere) seek to condense and package the experience and wisdom of these FLOSS practices into practical advice that may be offered to business executives considering when and why to adopt FLOSS options [13].

Many of these studies and insights about how FOSSD works result from reflective practice rather than systematic empirical study. However, personal reflection on software development practice often tends to (a) be uncritical with respect to prior scholarship or theoretical interpretation, or (b) employ unsystematic collection of data to substantiate pithy anecdotes or proffered conclusions. Thus, by themselves such studies offer a limited basis for further research or comparative analysis. Nonetheless, they can (and often do) offer keen insights and experience reports that may help sensitize future FOSSD researchers to interesting starting points or problems to further explore.

3. Understanding FLOSS development across different communities

It is best to assume there is no general model or globally accepted framework that defines how FLOSS is or should be developed. Subsequently, our starting point is to investigate FLOSS practices in different communities from an ethnographically informed, qualitative perspective [4, 25]. Here we briefly examine four different FOSSD communities to describe and compare. These are those centered about the development of software for networked computer games, Internet/Web infrastructure, scientific computing, and academic software engineering research. The following sections briefly introduce and characterize these FLOSS sub-domains. Along the way, example software systems or projects are *highlighted* or identified via external reference/citation, which can be consulted for further information or review.

3.1 Networked computer game worlds

Participants in this community focus on the development and evolution of first person shooters (FPS) games (e.g., *Quake Arena*, *Unreal Tournament*), massive multiplayer online role-playing games (e.g., *World of Warcraft, Lineage, EveOnline, City of Heroes*), and others (e.g., *The Sims* (Electronic Arts), *Grand Theft Auto* (Rockstar Games)). Interest in networked computer games and gaming environments, as well as their single-user counterparts, have exploded in recent years as a major mode of entertainment, playful fun, popular culture, and global computerization movement. The release of *DOOM*, an early first-person action game, onto the Web in open source form in the mid 1990's, began what is widely recognized the landmark event that launched the development and redistribution of computer game *mods* – open source and distributable modifications of commercially available games created with software development kits provided with the retail game package by the game's software developer. The end-user license agreement for games that allow for end-user created game mods often stipulate that the core game engine (or retail game software product) is protected as closed source, proprietary software that cannot be examined or redistributed, while any user created mod can only be redistributed as open source software that cannot be declared proprietary or sold outright, and must only be distributed in a manner where the retail game product must be owned by any end-user of a game mod. This has the effect of enabling a secondary market for retail game purchases by end-users or other game modders who are primarily interested in accessing, studying, playing, further modifying, and redistributing a game mod.

Mods are variants of proprietary (closed source) computer game engines that provide extension mechanisms like (domain-specific) game scripting languages (e.g., *UnrealScript* for mod development with *Unreal* game engines from Epic Games Inc.) that can be used to modify and extend a game. Extensions to a game created with these mechanisms are published for sharing across the Web with other game players, and are licensed for such distribution in an open source manner. Mods are created by small numbers of users who want and are able to modify games (they possess some software development skills), compared to the huge numbers of players that enthusiastically use the games as provided. The scope of mods has expanded to now include new game types, game character models and skins (surface textures), levels (game play arenas or virtual worlds), and artificially intelligent game bots (in-game opponents). For additional background on computer game mods, see http://en.wikipedia.org/wiki/Mod_(computer_gaming).

Perhaps the most widely known and successful game mod is *Counter-Strike* (CS), which is a total conversion of Valve Software's *Half-Life* computer game. Valve Software has since commercialized *CS* and many follow-on versions. CS was created by two game modders who were reasonably accomplished students of software development. Millions of copies of *CS* have subsequently been distributed, and millions of people have played *CS* over the Internet, according to http://counterstrikesource.net/. Other popular computer games that are frequent targets for modding include the *Quake, Unreal, Half-Life,* and *Crysis* game engines, *NeverWinter Nights* for role-playing games, motor racing simulation games (e.g., *GTR* series), and even the massively popular *World of Warcraft* (which only allows for modification of end-user interfaces, and not the game itself). Thousands of game mods are distributed through game mod portals like http://www.MODDB.com. In contrast, large successful game software companies like Electronic Arts and Microsoft do not embrace nor encourage game modding, and do not provide end-user license agreements that allow game modding and redistribution.

3.2 Internet/Web infrastructure

The SourceForge web portal (http://www.sourceforge.net), the largest associated with the FLOSS community, currently stores information on more than 1,750K registered users and developers, along with nearly 200K FOSSD projects (as of July 2008), with more than 10% of those projects indicating the availability of a mature, released, and actively supported software system. However, some of the most popular FLOSS projects have their own family of related projects, grouped within their own portals, such as for the Apache Foundation and Mozilla Foundation. Participants in this community focus on the development and evolution of systems like the *Apache* web server, *Mozilla/Firefox* Web browser, *GNOME* and *K Development Environment* (KDE) for end-user interfaces, the *Eclipse* and *NetBeans* interactive development environments for Java-based Web applications, and thousands of others. This community can be viewed as the one most typically considered in popular accounts of FLOSS projects. However, it is reasonable to note that the two main software systems that enabled the World Wide Web, the *NCSA Mosaic* Web browser (and its descendants, like Netscape Navigator, Mozilla, Firefox, and variants like *K-Meleon*, *Konqueror*, *SeaMonkey*, and others), and the Apache Web server (originally know as *httpd*) were originally and still active FOSSD projects.

The GNU/Linux operating system environment is one of the largest, most complex, and most diverse sub-community within this arena, so much so that it merits separate treatment and examination. Many other Internet or Web infrastructure projects constitute recognizable communities or sub-communities of practice. The software systems that are the focus generally are not standalone end-user applications, but are often directed toward *system administrators* or *software developers as the targeted user base,* rather than the eventual end-users of the resulting systems. However, notable exceptions like Web browsers, news readers, instant messaging, and graphic image manipulation programs are growing in number within the end-user community.

3.3 Scientific computing in X-ray astronomy and deep space imaging

Participants in this community focus on the development and evolution of software systems supporting the Chandra X-Ray Observatory, the European Space Agency's XMM-Newton Observatory, the Sloan Digital Sky Survey, and others. These are three highly visible astrophysics research projects whose scientific discoveries depend on processing remotely sensed data through a complex network of open source software applications that process remotely sensed data. In contrast to the preceding two development oriented FOSSD communities, open software plays a significant role in scientific research communities. For example, when scientific findings or discoveries resulting from remotely sensed observations are reported, then members of the relevant scientific community want to be assured that the results are not the byproduct of some questionable software calculation or opaque processing trick. In scientific fields like astrophysics that critically depend on software, open source is considered an essential precondition for research to proceed, and for scientific findings to be trusted and open to independent review and validation. Furthermore, as discoveries in the physics of deep space are made, this in turn often leads to modification, extension, and new versions of the astronomical software in use that enable astrophysical scientists to further explore and analyze newly observed phenomena, or to modify/add capabilities to how the remote sensing or astrophysical computation mechanisms operate. For example, the NEMO Stellar Dynamics Package at http://bima.astro.umd.edu/nemo/ is now at version 3.2.5, as of September 2008.

To help understand these matters, consider the example at http://antwrp.gsfc.nasa.gov/apod/ap010725.html .

This Website page displays a composite image constructed from both X-ray (Chandra Observatory) and optical (Hubble Space Telescope) sensors. The open software processing pipelines for each sensor are mostly distinct and are maintained by different organizations. However, their outputs must be integrated, and the images must be registered and oriented for synchronized overlay, pseudo-colored, and then composed into a final image, as shown on the cited Web page. Subsequently, there are dozens of open software programs that must be brought into alignment for such an image to be produced, and for such a scientific discovery to be claimed and substantiated.

3.4 Academic software systems design

Participants in this community focus on the development and evolution of software architecture and UML centered design efforts, such as for ArgoUML (http://argouml.tigris.org) or xARCH at CMU and UCI (http://www.isr.uci.edu/projects/xarch/). This community can easily be associated with a mainstream of SE research. People who participate in this community generally develop software for academic research or teaching purposes in order to explore topics like software design, software architecture, software design modeling notations, software design recovery (reverse software engineering), etc. Accordingly, it may not be unreasonable to expect that open software developed in this community should embody or demonstrate principles from modern SE theory or practice. Furthermore, much like the X-ray astronomy community, members of this community expect that when breakthrough technologies or innovations have been declared, such as in a refereed conference paper or publication in a scholarly journal, the opportunity exists for other community members to be able to access, review, or try out the software to assess and demonstrate its capabilities. An inability to provide such access may result in the software being labeled as "vaporware" and the innovation claim challenged, discounted, or rebuked. Alternatively, declarations of "non-disclosure" or "proprietary intellectual property" are generally not made for academic software, unless or until it is transferred to a commercial firm. However, it is often acceptable to find that academic software constitutes nothing more than a "proof of concept" demonstration or prototype system, not intended for routine or production use by end-users.

3.5 Overall characteristics across different FOSSD communities

In contrast to efforts that draw attention to generally one (but sometimes many) open source development project(s) within a single community [e.g., 4, 7, 12, 18, 23], there is something to be gained by examining and comparing the communities, processes, and practices of FOSSD in different communities. This may help clarify what observations may be specific to a given community (e.g., GNU/Linux projects), compared to those that span multiple, and mostly distinct communities. Two of the communities identified above are primarily oriented to develop software to support scholarly research (scientific computing and academic software systems design) with rather small user communities. In contrast, the other two communities are oriented primarily towards software development efforts that may replace/create commercially viable systems that are used by large end-user communities. Thus, rather than focusing on a single community as the primary source for insight, reflective practice, or systematic empirical study, it is clear that FOSSD processes, practices, and projects can be studied with or across (a) disparate communities, as well as (b) different application domains.

Each of these highlighted items (in italics or via Web links above) point to the public availability of data that can be collected, analyzed, and re-represented within ethnographic narratives [4, 7], computational process models [24], or for quantitative studies [16, 17]. Significant examples of each kind of data (source code and artifacts, development processes, project repositories, etc.) have been collected and analyzed by different FOSSD researchers [11]. A later section highlights different types of FOSSD studies that use such substantial, diverse, and publicly available data. The next section identifies a number of FOSSD artifact types that serve to document the social actions and technical practices that facilitate and constrain FOSSD processes.

Beyond the communities identified above, other empirical studies of FOSSD reveal that FOSSD work practices, engineering processes, and project community dynamics can best be understood through observation and examination of their socio-technical elements at different, or across multiple, levels of analysis. In particular, FOSSD projects can be examined through a *micro-level* analysis of (a) the actions, beliefs, and motivations of individual FOSSD project participants [4, 7], and (b) the social or technical resources that are mobilized and configured to support, subsidize, and sustain FOSSD work and outcomes [12, 18, 22]. Similarly, FOSSD projects can be examined through *meso-level* analysis of (c) patterns of cooperation, coordination, control, leadership, role migration, and conflict mitigation [2, 7, 15, 22], and (d) project alliances and inter-project socio-technical networking [14, 16, 23]. Last, FOSSD projects can also be examined through *macro-level* analysis of (d) multi-project FLOSS ecosystems [17], and (e) FOSSD as a social movement and emerging global culture [6]. Each of these levels of analysis can be decomposed into finer granularity topics for study. For example, micro-level studies can address topics in FOSSD including (1) motives or incentives for individual participation, (2) use of personal computing resources, (3) individual beliefs supporting free versus open source software development, (4) self-organization and self-management of competent FLOSS developers, (5) the discretionary time and effort committed by FLOSS developers, (6) trust and social accountability mechanisms, or (7) types and uses for different types of online artifacts that support decentralized development work of project participants. We take this last topic for further elaboration in the next section. However, it should be clear that studies of FOSSD at any level of analysis will require appropriately selected research methods and data samples [11].

4. Informalisms for describing FLOSS requirements and design rationale

Functional and non-functional requirements for FLOSS systems are elicited, analyzed, specified, validated, and managed through a variety of Web-based artifacts. These descriptive documents can be treated as software informalisms. *Software informalisms* [22] are the information resources and artifacts that participants use to describe, proscribe, or prescribe what's happening in a FOSSD project. They are informal narrative resources codified in lean descriptions that are comparatively easy to use, and publicly accessible to those who want to join the project, or just browse around. An early study demonstrated how software informalisms can take the place of formalisms, like "requirement specifications" or software design notations which are documentary artifacts seen as necessary to develop high quality software according to the SE community [22]. Yet these software informalisms often capture the detailed rationale, contextualized discourse, and inquisitive debates for why changes were made in particular development activities, artifacts, or source code files. The example in Exhibit 1 displays a software informalism in the form of a threaded message discussion that describes, contextualizes, and rationalizes why a new software system version and feature set were implemented and now required. The contents of such an informalism embodies requires peer review and comprehension by other developers on a project before acceptable or trusted contributions can be made, or retrospectively justified [7, 15]. Finally, the choice to designate these descriptions as informalisms is to draw a distinction between how the requirements of FLOSS systems are described, in contrast to the recommended use of formal requirements notations and deliverable documents ("formalisms") that are advocated in traditional approaches [26]. However, to be clear, though SE often stresses the need to formalize software requirements specifications and designs, formalisms need not be limited to those based on a mathematical logic or state transition semantics, but can include descriptive schemes that are formed from structured or semi-structured narratives, such as those employed in Software Requirements Specifications documents. But as described below, FOSSD informalisms are generally quite different from formal SE documents.

--------------- Exhibit 1 about here -------------

Two dozen types of software informalisms can be identified, and each has sub-types that can be further identified and studied more closely. Each FOSSD project usually employs only a subset as its informal document ecosystem that meets the interests or needs of local project participants. There are no guidelines or best practices for which informalisms to use for what. Instead, we have observed practices that recur across FOSSD projects. Thus it is pre-mature and perhaps inappropriate to seek to further organize these informalisms into a classification or taxonomic scheme whose purpose is to prescribe when or where best to use one or another. Subsequently, they are presented here as an unordered list since to do so otherwise would transform this descriptive information into an interpretive scheme that suggests untested, hypothetical prescriptions [22].

The most common informalisms used in FOSSD projects include (i) communications and messages within project Email, (ii) threaded message discussion forums (see Exhibit 1), bulletin boards, or group blogs, (iii) news postings, and (iv) instant messaging or Internet relay chat. These enable developers and users to converse with one another in a lightweight, semi-structured manner, and now use of these tools is global across applications domains and cultures. As such, the discourse captured in these tools is a frequent source of FLOSS requirements. A handful of FOSSD projects have found that summarizing these communications into (v) project digests [7] helps provide an overview of major development activities, problems, goals, or debates. A project digest, such as that displayed in Exhibit 2, represents a multi-participant summary that records and hyperlinks the original contextualized rationale accounting for focal project activities, development problems, current software quality status, and desired software functionality. Project digests (which sometimes are identified as “kernel cousins”) record the discussion, debate, consideration of alternatives, code patches and initial operational/test results drawn from discussion forums, online chat transcripts, and related online artifacts [7].

--------------- Exhibit 2 about here -------------

As FLOSS developers and users employ these informalisms, they have been found to also serve as carriers of technical beliefs and debates over desirable software features, social values (e.g., reciprocity, freedom of choice, freedom of expression), project community norms, as well as affiliation with the global FLOSS social movement [6, 10].

Other common informalisms include (vi) scenarios of usage as linked Web pages or screenshots of software in operation, (vii) how-to guides, (viii) to-do lists, (ix) Frequently Asked Questions, and other itemized lists, and (x) project Wikis, as well as (xi) traditional system documentation and (xii) external FOSSD publications [3, 9]. FLOSS (xiii) project property licenses (whether to assert collective ownership, transfer copyrights, insure “copyleft,” or some other reciprocal agreement) are documents that also help to define what software or related project content are protected resources that can subsequently be shared, examined, modified, and redistributed.

Next, (xiv) open software architecture diagrams, (xv) intra-application functionality realized via scripting languages like Perl or PhP, and the ability to either (xvi) incorporate externally developed software modules or “plug-ins”, or (xvii) integrate software modules from other FOSSD efforts, are all resources that are used informally, where or when needed according to the interests or actions of project participants.

All of the software informalisms are found or accessed from (xix) project related Web sites or portals. These Web environments are where most FLOSS software informalisms can be found, accessed, studied, modified, cross-referenced and redistributed.

A Web presence helps make visible the project's information infrastructure and the array of information resources that populate it. These include FOSSD multi-project Web sites (e.g., SourgeForge.net, Savanah.org, Freshment.org, Tigris.org, Apache.org, Mozilla.org, or Google.com/code), community software Web sites (PhP-Nuke.org), and project-specific Web sites (e.g., www.GNUenterprise.org), as well as (xx) embedded project source code Webs (directories), (xxi) project repositories such as those implemented using CVS or Subversion [9], and (xxii) software bug reports and (xxiii) issue tracking data base like Bugzilla [see http://www.bugzilla.org/]. Last, giving the growing global interest in online social networking, it not surprising to find increased attention to documenting various kinds of social gatherings and meetings using (xxiv) social media Web sites (e.g, YouTube, Google Video, Flickr, MySpace, etc.) where FLOSS developers, users, and interested others come together to discuss, debate, or work on FLOSS projects, and to use these online media to record, and publish photographs/videos that establish group identity and affiliation with different FLOSS projects. For example, a video presentation from 2007, http://video.google.com/videoplay?docid=-4216011961522818645 (accessed September 2008), describes how to identify and remove troublesome or unwelcome members of a FOSSD project team, who may cause the project to fragment, lose focus, become conflict laden, or otherwise fail.

In FOSSD projects, software informalisms are the preferred scheme for describing or representing FLOSS requirements, rationalizing software design choices, discussing alternative software implementations and releases, and more. There is no explicit objective or effort to treat these informalisms as "informal software requirements" that should be refined into formal requirements or design notations within these communities (academic software design community perhaps being the exception). Accordingly, each of the available types of FOSSD informalisms have been found in one or more of the four communities identified above. Subsequently, knowledge about who is doing what, where, when, why, and how is captured in different or multiple informalisms.

Together, these two dozen types of software informalisms constitute a substantial yet continually evolving web of informal, semi-structured, or processable information resources that capture, organize, and distribute knowledge that embody the requirements for an FOSSD project. This web results from the hyperlinking and cross-referencing that interrelate the contents of different informalisms together. Subsequently, these FLOSS informalisms are produced, used, consumed, or reused within and across FOSSD projects. They also serve to act as both a distributed virtual repository of FLOSS project assets, as well as the continually adapted distributed knowledge base through which project participants evolve what they know about the software systems they develop and use.

Overall, it appears that none of these software informalisms would defy an effort to formalize them in some mathematical logic or analytically rigorous notation. Nonetheless, in the three of the four software communities examined in this study, there is no perceived requirement for such formalization (academic software design being the exception), as the basis to improve the quality, usability, or cost-effectiveness of the FLOSS systems. If formalization of these documentary software informalisms has demonstrable benefit to members of the other three communities, beyond what they already realize from current practices, these benefits have yet to be articulated in the different types of software informalisms for online discourse that pervades each community. However, in contrast, the academic software design community often does encourage and embrace the development of formal requirements specification and design documents in order to coordinate and guide development of their software projects. Thus, as before, understanding FOSSD practices, processes, or projects can vary by community, such that studying such projects or associated artifacts in only one community can skew or bias what can be observed or put into practice when comparing across multiple communities. This in turn sets the stage for examining how studies of FOSSD may be organized or structured to facilitate comparative investigations.

5. FOSSD Research Studies and Approaches

A growing group of software researchers is beginning to investigate large software projects empirically, using freely available data from FLOSS projects. A body of recent work point out the need for community-wide data collections and research infrastructure to expand the depth and breadth of empirical FLOSS research, and several initial proposals have been made. Most importantly, these data collections [16, 17] and shared research infrastructure [14] are intended to support an active and growing community of FLOSS scholars addressing contemporary issues and theoretical foundations in disciplines that include anthropology, economics, informatics (computer-supported cooperative work), information systems, library and information science, management of technology and innovation, law, organization science, policy science, and software engineering [11]. Approximately 700 published studies can be found at the FLOSShub collection of research papers, at http://flosshub.org/biblio. Furthermore, this research community has researchers based in Asia, Europe, North America, South America, and the South Pacific, thus denoting its international and global membership. Consequently, the research community engaged in empirical studies of FLOSS can be recognized as another member in the growing movement for interdisciplinary software engineering research.

This section seeks to clarify the need for community-wide, sharable research infrastructure and collections of FLOSS data. The kinds of research questions that such data sampling and corresponding research methods enable are discussed elsewhere [11].

5.1 Objects of study, success indicators, and critical factors for understanding FOSSD

As an organizing framework, there are primarily five main *objects of study*--that is, things whose characteristics researchers are trying to describe and explain--in F/OSS-based empirical software research: *software artifacts and source code*, *software processes*, *development projects, communities*, and *participants' knowledge*. Table 1 provides a simplified set of associations between representative characteristics of FOSSD that have been investigated for each of these objects of study, and show some critical factors that researchers have begun linking to these characteristics as explanations. It is important to point out that these objects of study are by no means independent from one another. They should be considered as interdependent elements of FLOSS (e.g., knowledge and processes affect artifacts, communities affect processes, etc.). Also, each of the indicators of success shown in Table 1 may play a role as a critical factor in other categories.

5.2 Current FOSSD research approaches

Based on published studies of FOSSD since 2000, there are four alternative approaches in empirical research on the objects of study, measures of variables that contribute to FOSSD success, and causal mechanisms or factors that drive FOSSD success, as shown in Table 1 [11]:

Objects	Success Measures	Critical Driving Factors
Source Code and Artifacts	Quality, downloads, reliability, usability, durability, fit, structure, growth, diversity, localization	Size, complexity, bugs and features, software architecture (structure, substrates, modularity, versions, infrastructure), information architecture, artifact types, and document genres
Processes	Efficiency, ease of improvement, adaptability, effectiveness, complexity, manageability, predictability	Size, distribution, collaboration, knowledge/information management, artifact structure, configuration, agility, innovativeness
Projects	Type, size, duration, number of participants, number of software versions released	Development platforms, tools supporting development and project coordination, software imported from elsewhere, social networks, roles and role migration paths, leadership and core developers, socio-technical vision
Communities	Ease of creation, sustainability, trust, increased social capital, rate of participant turnover	Size, economic setting, organizational architecture, behaviors, incentive structures, institutional forms, motivation, participation, core values, common-pool resources, public goods
Knowledge	Creation, codification, use, need, management	Tools, conventions, norms, social structures, technical content, acquisition, representations, reproduction, application

Table 1: Characteristics of empirical FLOSS studies.

- *Very large, population-scale studies* examining common objects selected and extracted from hundreds to tens-of-thousands of FLOSS projects from multi-project repositories or FLOSS research portals like FLOSSmole [14], or from international surveys of comparable numbers of FLOSS developers.
- *Large-scale cross-analyses of project and artifact characteristics*, such as code size and code change evolution, development group size, composition and organization, or development processes [16, 17].
- *Medium-scale comparative studies* across multiple FLOSS projects within different communities or software system application domains [15, 17, 18].
- *Smaller-scale in-depth case studies* of specific FLOSS practices and processes, for concept or

hypothesis development, and for exposing and investigating details of socio-technical interactions [4, 7, 12, 18].

These four alternative research strategies are separated less by fundamental differences in objectives than by technical limitations in existing tools and research methods, or by the socio-technical research constraints associated with quantitative studies of data mined from publicly accessible repositories versus the challenges of qualitative ethnographic research methods. For example, qualitative analyses are hard to implement on a large scale, while quantitative methods must rely on uniform, easily processed data. We believe these distinctions are becoming increasingly blurred as researchers develop and use more sophisticated analysis and modeling tools [25], leading to finer gradations in empirical data needs. Accordingly, attention can now turn to identifying opportunities for future research and practice that may span FOSSD and SE.

6. OPPORTUNITIES FOR FOSSD AND SE RESEARCH AND PRACTICE

There are a significant number of opportunities and challenges that arise when we look to identifying which software development or socio-technical interaction practices found in studies of FOSSD projects might be applied in the world of SE. Some of these opportunities follow. However, it is perhaps surprising to observe that it is unclear whether the world of FOSSD is interested in adopting the best practices found in the world of SE, and vice versa. For example, why would software developers seek out to engage in FOSSD projects and practices if they were the same or less than those found in SE? That is, most FOSSD projects are not seen as stepping stones to SE, nor feeble attempts at SE. Instead, FOSSD projects in most communities are seen as currently the most effective way to develop and sustain both large software systems as well as a community of like-minded developers and end-users. As such, let us consider the opportunities for what FOSSD might contribute to the world SE.

First, FLOSS poses the opportunity to favorably alter the costs and constraints of accessing, analyzing, and sharing software process and product data, metrics, and data collection instruments. FOSSD is thus poised to alter the calculus of empirical SE. Software process discovery, modeling, and simulation research [25] is one arena that can take advantage of such a historically new opportunity. Similarly, the ability to extract or data mine software product content (source code, development artifacts, etc.) within or across FLOSS project repositories to support its visualization, restructuring/refactoring, or redesign has become a high-yield, high impact area for SE study and experimentation. Another would be examining the effectiveness and efficiency of traditional face-to-face-to-artifact SE approaches or processes for software inspections compared to the online peer reviews and multi-modal discourse informalisms prevalent in FOSSD efforts.

Second, based on results from studies of motivation, participation, role migration, and work practices of individual FLOSS developers [4, 15], it appears that the SE community would benefit from empirical studies that examine similar conditions in conventional or proprietary software development enterprises. Current SE textbooks and development processes seem to assume that individual developers have simple technical roles (e.g., software designer or programmer) and motivations driven by financial compensation and technical education, and therefore seek the quality assuring rigor that purportedly follows from the use of formal notations and mathematically based analytical schemes. Consequently, if FOSSD is more fun, interesting, and rewarding compared to SE, then what can be done to make SE more fun, interesting and rewarding to SE students and new software developers? Said simply, SE may benefit from FOSSD practices that developers find are more fun, interesting, and rewarding, rather than merely following the

40 year legacy of SE practices, processes, and procedures prescribed in SE textbooks.

Third, based on results from studies of resources and capabilities employed to support FOSSD projects [7, 12, 18, 22], it appears that conventional software cost estimation or accounting techniques (e.g., "total cost of operation" or TCO) are limited to analyzing resources or capabilities that are easily quantified or monetized. This in turns suggests that many social and organizational resources/capabilities are slighted or ignored, thus producing results that miscalculate the diversity of resources and capabilities that affect the ongoing/total costs of software development projects, whether FLOSS or SE based.

Fourth, based on results from studies of cooperation, coordination, and control in FOSSD projects [2, 12, 15, 23], it appears that virtual project management and socio-technical role migration/advancement can provide a slimmer and lighter weight approach to SE project management. However, it is unclear whether we will see corporate experiments in SE that choose to eschew traditional project management and administrative control regimes in favor of enabling software developers the freedom of choice and expression that may be necessary to help provide the intrinsic motivation to self-organize and self-manage their SE project work.

Fifth, based on results of studies on alliance formation, inter-project social networking, community development, and multi-project software ecosystems [14, 16, 17], it appears that SE projects currently operate at a disadvantage compared to FOSSD projects. In SE projects, it is commonly assumed that developers and end-users are distinct communities, and that ongoing software evolution is governed by market imperatives, the need to extract maximum marginal gains (profit), and resource-limited software maintenance effort. SE efforts are setup to produce systems whose growth and evolution is limited, rather than capable of sustaining exponential growth of co-evolving software functional capability and developer-user community [23].

Last, based on studies of FOSSD as a social movement [6], it appears that there is an opportunity and challenge for encouraging the emergence of a social movement that combines the best practices of FOSSD and SE. The emerging niche world of open source software engineering (OSSE) is the likely locus of collective action that might enable such a movement to arise. For example, as suggested in Exhibit 3, the community Web portal for Tigris.org is focused on cultivating and nurturing the emerging OSSE community—the community that spans both FOSSD and SE. Nearly 1000 OSSE projects are currently affiliated with this portal and community. It might therefore prove fruitful to closely examine different samples of OSSE projects at Tigris.org to see which SE tools, techniques, and concepts are being brought to bear and to what ends in different FLOSS projects, as well as which SE concepts and methods are employed to develop FLOSS-based software development tools.

----------- Insert Exhibit 3 about here -------------

7. Limitations and Conclusions on FOSSD

FOSSD is certainly not a panacea for developing complex software systems, nor is it simply SE done poorly. Instead, it represents an alternative community-intensive socio-technical approach to develop software systems, artifacts, and social relationships. However, it is not without its limitations and constraints. Thus, we should be able to help see these limits as manifest within the level of analysis or research for empirical FOSSD studies examined above.

First, in terms of participating, joining, and contributing to FOSSD projects, an individual developer's interest, motivation, and commitment to a project and its contributors is dynamic and not indefinite. Some form of reciprocity and self-serving or intrinsic motivation seems necessary to sustain participation in a FOSSD project, whereas a perception of exploitation by others can quickly dissolve a participant's commitment to further contribute, or worse to dissuade other participants to abandon a FOSSD project that has gone astray. Similarly, most FOSSD projects are not open to anyone who just wants to immediately become a core developer, system architect, or group leader without going through a protracted socio-technical process that requires demonstration of technical competence, as well as willingness to help other project participants and facilitate project growth in ways aligned to the dominant interests of the overall project community. Developers who want to pursue alternative approaches, software architectures, or divisions of labor may be undesirable, unwanted, and at times be pushed out of a FOSSD project (cf. Footnote 8). FOSSD is as much about developing and sustaining the project community as it is about the software and how it is being developed.

Second, in terms of cooperation, coordination, and control, FOSSD projects do not escape conflicts in technical decision-making, or in choices of who gets to work on what, or who gets to modify and update what. As FOSSD projects generally lack traditional project managers, then they must become self-reliant in their ability to mitigate and resolve outstanding conflicts and disagreements. Beliefs and values that shape system design choices, as well as choices over which software tools to use, and which software artifacts to produce or use, are determined through negotiation rather than administrative assignment. Negotiation and conflict management then become part of the cost that FLOSS developers must bear in order for them to have their beliefs and values fulfilled. It is also part of the cost they bear in convincing and negotiating with others often through electronic communications to adopt their beliefs and values. Time, effort, and attention spent in negotiation and conflict management represent an investment in building and sustaining a negotiated socio-technical network of dependencies.

Third, in terms of forming alliances and building community through participation, artifacts, and tools points to a growing dependence on other FOSSD projects. The emergence of non-profit foundations that were established to protect the property rights of large multi-component FOSSD project creates a demand to sustain and protect such foundations. If a foundation becomes too bureaucratic, then this may drive contributors away from a project. So, these foundations need to stay lean, and not become a source of occupational careers, in order to survive and evolve. Similarly, as FOSSD projects give rise to new types of requirements for community building, community software, and community information sharing systems, these requirements need to be addressed and managed by FOSSD project contributors in roles above and beyond those involved in enhancing the source code of a FOSSD project. FOSSD alliances and communities depend on a rich and growing web of socio-technical relations. Thus, if such a web begins to come apart, or if the new requirements cannot be embraced and satisfied, then the FOSSD project community and its alliances will begin to come apart.

Fourth, in terms of the co-evolution of FLOSS systems and community, as already noted, individual and shared resources of people's time, effort, attention, skill, sentiment (beliefs and values), and computing resources are part of the socio-technical web of FOSS. Reinventing existing software systems as FLOSS coincides with the emergence or reinvention of a community who seeks to make such system reinvention occur. FLOSS systems are "common pool resources" that require collective action for their development, mobilization, use, and evolution. Without the collective action of the FOSSD project community, the common pool will dry up, and without the common pool, the community begins to fragment, drift away and disappear, perhaps to search for another pool elsewhere.

Last, empirical studies of FOSSD are expanding the scope of what we can observer, discover, analyze, or learn about how large software systems can be or have been developed. In addition to traditional methods used to investigate FOSSD like reflective practice, industry polls, survey research, and ethnographic studies, comparatively new techniques for mining software repositories and multi-modal modeling and analysis of the socio-technical processes and networks found in sustained FOSSD projects [7, 15, 24] show that the empirical study of FOSSD is growing and expanding. This in turn will contribute to and help advance the empirical science in fields like SE, which previously were limited by restricted access to data characterizing large, proprietary software development projects. Thus, the future of empirical studies of software engineering practices, processes, and projects will increasingly be cast as studies of free/open source software development efforts.

Overall, one of the defining characteristics of FOSSD projects is that data about development processes, work practices, and project/community dynamics is publicly available on a global basis. Data about FOSSD products, artifacts, and other resources is kept in repositories associated with a project's Web site, though finding and extracting it may involve substantial effort. This may include the site's content management system, computer mediated communication systems (email, persistent chat facilities, and discussion forums), software versioning or configuration management systems, and networked file systems. FOSSD process data is generally either extractable or derivable from data/content in these artifact repositories, though not always easily, nor in forms readily amenable to systematic analysis without further processing, reformatting and cleaning. First-person data may also be available to those researchers or students who participate in a project, even if just to remotely observe ("lurk") or to electronically interview other participants about development activities, tools being used, the status of certain artifacts, and the like. The availability of such data suggest a growing share of empirical SE research will henceforth be performed in domains of FOSSD projects, rather than using data from in-house or proprietary software development projects that have constraints on access and publication. FOSSD process data collection from publicly accessible artifact repositories will be found to be more cost-effective compared to studies of traditional closed-source, proprietary, and in-house software development repositories [11]. Your chance to further study or participate in a FOSSD project starts from here.

Acknowledgements

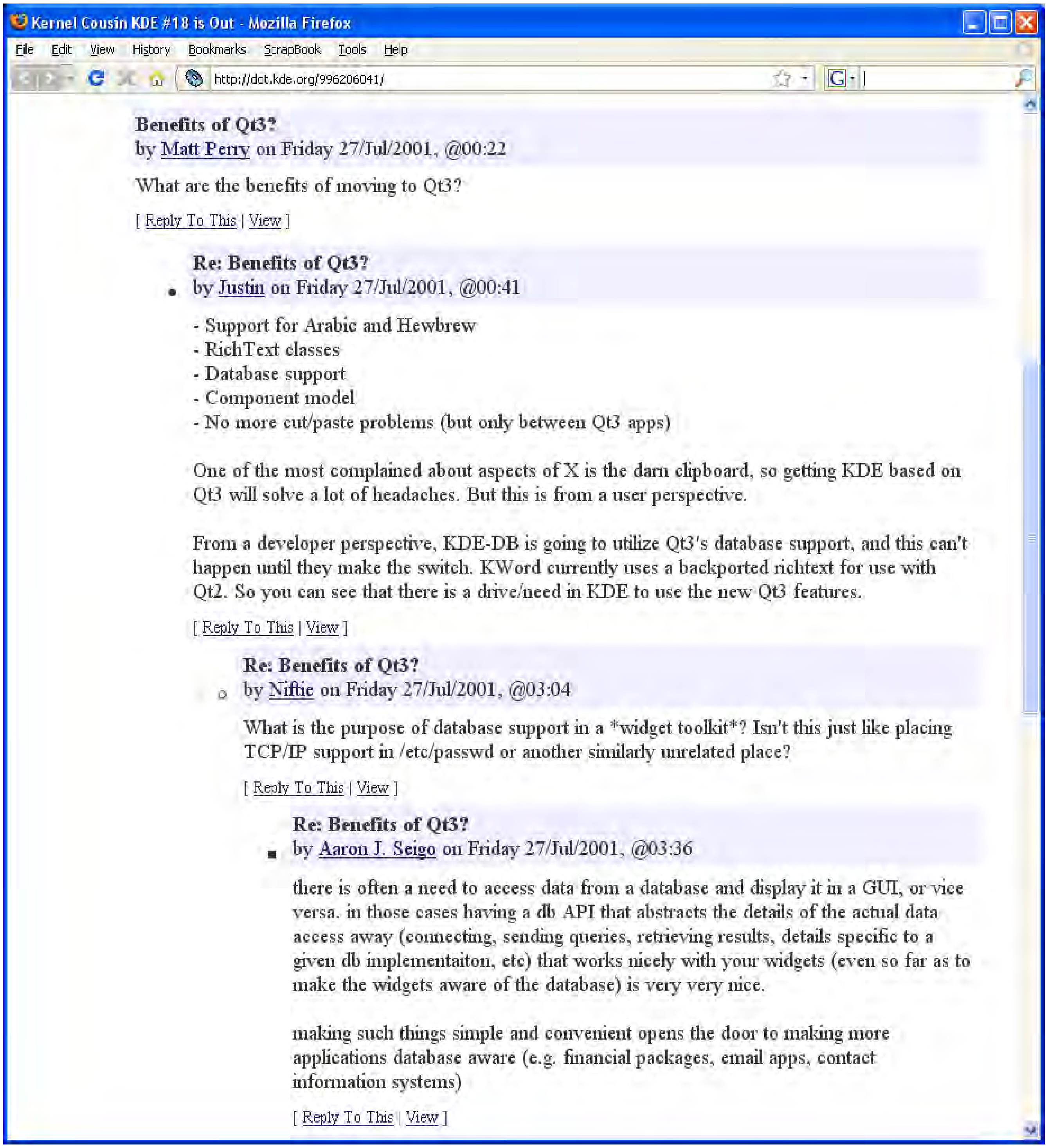
Research for this article was supported by grants #0534771, #0749353, and #0808783 from the U.S. National Science Foundation; and from the Acquisition Research Program, and also the Center for the Edge, at the Naval Postgraduate School, Monterey, CA.

References

1. Cambell-Kelly, M. and Garcia-Swartz, D.D., Pragmatism, not ideology: Historical perspectives IBM's adoption of open-source software, *Information Economics and Policy*, 2009, 21(3), 229-244.
2. Crowston, K., and Scozzi, B., Open Source Software Projects as Virtual Organizations: Competency Rallying for Software Development, *IEE Proceedings--Software*, 2002, 149(1), 3-17.
3. DiBona, C., Cooper, D., and Stone, M. (Eds.), *Open Sources 2.0*, 2005, O'Reilly Media, Sebastopol, CA. Also see, C. DiBona, S. Ockman, and M. Stone (Eds.). *Open Sources: Vocides from the Open Source Revolution*, 1999. O'Reilly Media, Sebastopol, CA.
4. Ducheneaut, N. Socialization in an Open Source software community: A socio-technical analysis. *Computer Supported Cooperative Work*, 2005, 14(4), 323-368.

5. Eliot, L. and Scacchi, W., Developing a Knowledge-Based System Factory: Issues and Concepts, *IEEE Expert*, 1986, 1(4), 51-58.
6. Elliott, M., Examining the Success of Computerization Movements in the Ubiquitous Computing Era: Free and Open Source Software Movements, in M. Elliott and K.L. Kraemer (Eds.), *Computerization Movements and Technology Diffusion: From Mainframes to Ubiquitous Computing*, Information Today, Inc, Medford, NJ, 2008, 359-380.
7. Elliott, M., Ackerman, M.S., and Scacchi, W.,Knowledge Work Artifacts: Kernel Cousins for Free/Open Source Software Development, *Proc. ACM Conf. Support Group Work (Group07)*, Sanibel Island, FL, ACM Press, 2007, 177-186, November.
8. Feller, J., Fiztgerald, B., Hissam, S. and Lakhani, K. (eds.), *Perspectives on Free and Open Source Software*, 2005, MIT Press, Cambridge, MA.
9. Fogel, K. *Producing Open Source Software: How to Run a Successful Free Software Project*, 2005, O'Reilly Press, Sebastopol, CA.
10. Gay, J. (ed.), *Free Software Free Society: Selected Essays of Richard M. Stallman*, 2005, GNU Press, Free Software Foundation, Boston, MA.
11. Gasser, L. and Scacchi, W., Towards a Global Research Infrastructure for Multidisciplinary Study of Free/Open Source Software Development, in IFIP Intern. Federation Info. Processing, Vol. 275; *Open Source Development, Community and Quality*; B. Russo, E. Damiani, S. Hissan, B. Lundell, and G. Succi (Eds.), Boston, Springer, 2008, 143-158.
12. German, D., The GNOME Project: A case study of open source, global software development, *Software Process—Improvement and Practice*, 2003, 8(4), 201-215.
13. Goldman, R. and Gabriel, R.P., *Innovation Happens Elsewhere: Open Source as Business Strategy*, Morgan Kaufmann Publishers, San Francisco, CA, 2005.
14. Howison, J., Conklin, M., and Crowston, K., FLOSSmole: A Collaborative Repository for FLOSS Research Data and Analyses. *Intern. J. Info. Tech. And Web Engineering*, 2006, 1(3), 17-26.
15. Jensen, C. and Scacchi, W., Role Migration and Advancement Processes in OSSD Projects; A Comparative Case Study, *Proc. 29th. Intern. Conf. Software Engineering*, Minneapolis, MN, ACM Press, 2007, 364-374.
16. Lopez-Fernandez, L., Robles, G., Gonzalez-Barahona, J.M., and Herraiz, I., Applying Social Network Analysis to Community-Drive Libre Software Projects, *Intern. J. Info. Tech. and Web Engineering*, 2006, 1(3), 27-28.
17. Madey, G., Freeh, V., and Tynan, R., Modeling the F/OSS Community: A Quantitative Investigation, in S. Koch (ed.), *Free/Open Source Software Development*, IGI Publishing, Hershey, PA, 2005, 203-221.
18. Mockus, A., Fielding, R., & Herbsleb, J.D., Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology*, 2002, 11(3), 309-346.
19. Moore, J.T.S., *Revolution OS*, 2001, See http://www.revolution-os.com/ Also available for viewing at *Google Videos* and the *Internet Archive.*
20. Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., *Numerical Recipes: The Art of Scientific Computation*, 1986, Cambridge University Press, NewYork. Also see, W.T. Vetterling, W.T., Teukolsky, S.A., Press, W.H. and Flannery, B.P., *Numerical Recipes Example Book (FORTRAN)*, 1985, Cambridge University Press, NewYork.
21. Scacchi, W., The Software Infrastructure for a Distributed System Factory, *Software Engineering Journal*, IEE and British Computer Society, 1991, 6(5), 355-369.
22. Scacchi, W., Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 2002, 149(1), 24-39, February.

23. Scacchi, W., Free/Open Source Software Development Practices in the Computer Game Community, *IEEE Software*, 2004, 21(1), 59-67, January/February.
24. Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S. and Lakhani, K, Understanding Free/Open Source Software Development Processes, *Software Process--Improvement and Practice*, 2006, 11(2), 95-105, March/April.
25. Scacchi, W., Jensen, C., Noll, J., and Elliott, M. Multi-Modal Modeling, Analysis, and Validation of Open Source Software Development Processes, *Intern. J. Internet Technology and Web Engineering,* 1(3), 49-63, 2006.
26. Sommerville, I., *Software Engineering*, 8th Edition, 2006, Addison-Wesley, New York.
27. Spencer, D.D., *Game Playing With Computers*, 1968, Spartan Books, New York.


Kernel Cousin KDE #18 is Out - Mozilla Firefox
File Edit View History Bookmarks ScrapBook Tools Help
http://dot.kde.org/996206041/

Benefits of Qt3?
by Matt Perry on Friday 27/Jul/2001, @00:22

What are the benefits of moving to Qt3?

[Reply To This | View]

Re: Benefits of Qt3?
by Justin on Friday 27/Jul/2001, @00:41

- Support for Arabic and Hewbrew
- RichText classes
- Database support
- Component model
- No more cut/paste problems (but only between Qt3 apps)

One of the most complained about aspects of X is the darn clipboard, so getting KDE based on Qt3 will solve a lot of headaches. But this is from a user perspective.

From a developer perspective, KDE-DB is going to utilize Qt3's database support, and this can't happen until they make the switch. KWord currently uses a backported richtext for use with Qt2. So you can see that there is a drive/need in KDE to use the new Qt3 features.

[Reply To This | View]

Re: Benefits of Qt3?
by Niftie on Friday 27/Jul/2001, @03:04

What is the purpose of database support in a *widget toolkit*? Isn't this just like placing TCP/IP support in /etc/passwd or another similarly unrelated place?

[Reply To This | View]

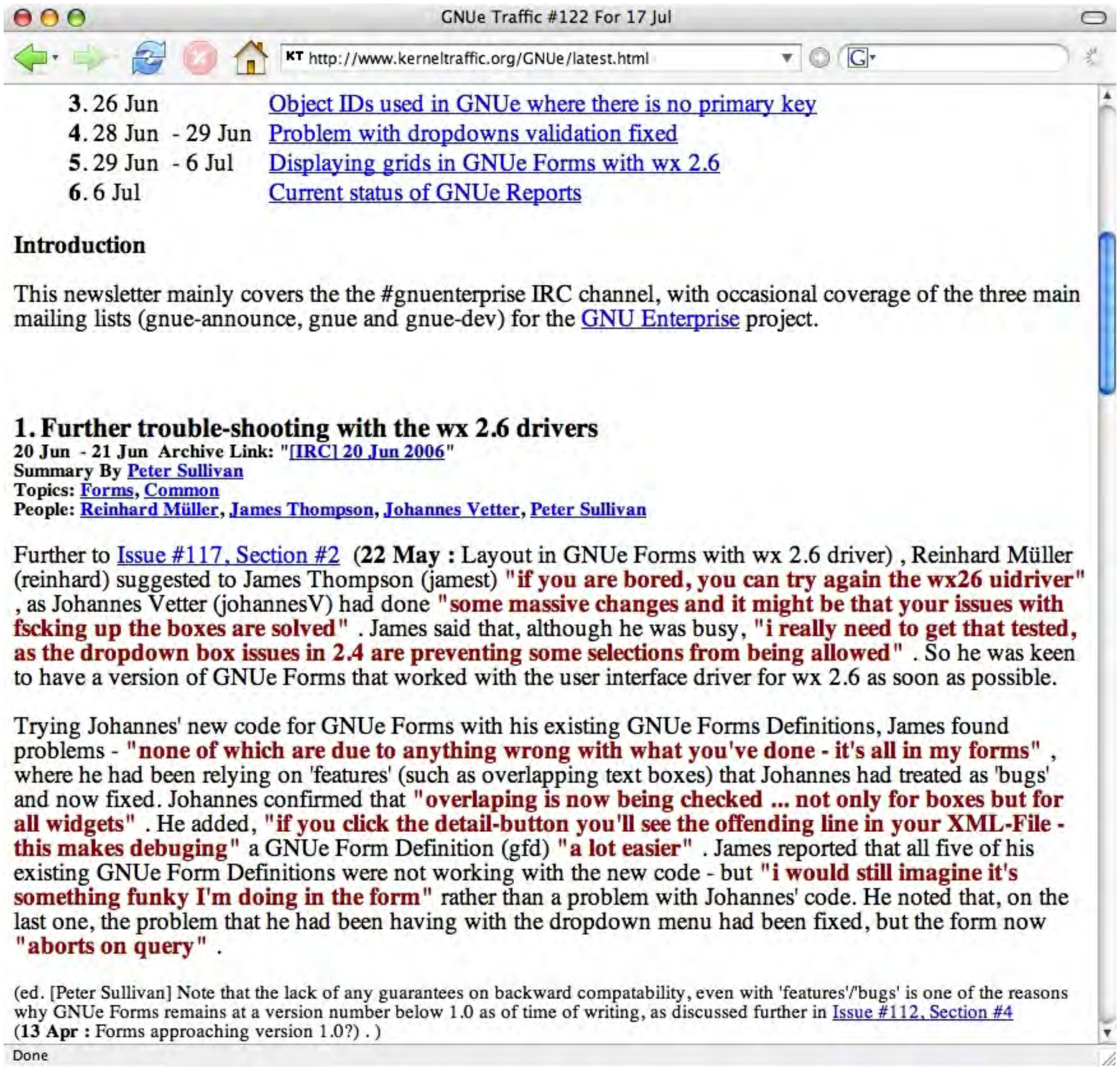
Re: Benefits of Qt3?
by Aaron J. Seigo on Friday 27/Jul/2001, @03:36

there is often a need to access data from a database and display it in a GUI, or vice versa. in those cases having a db API that abstracts the details of the actual data access away (connecting, sending queries, retrieving results, details specific to a given db implementaiton, etc) that works nicely with your widgets (even so far as to make the widgets aware of the database) is very very nice.

making such things simple and convenient opens the door to making more applications database aware (e.g. financial packages, email apps, contact information systems)

[Reply To This | View]

Exhibit 1. A screenshot display of a software informalism use to described, contextualize, and rationalize why a particular software feature was implemented and is thus required (Source: http://dot.kde.org/996206041/, accessed October 2008. Also appears in [22]).



GNUe Traffic #122 For 17 Jul

http://www.kerneltraffic.org/GNUe/latest.html

3. 26 Jun — Object IDs used in GNUe where there is no primary key
4. 28 Jun - 29 Jun — Problem with dropdowns validation fixed
5. 29 Jun - 6 Jul — Displaying grids in GNUe Forms with wx 2.6
6. 6 Jul — Current status of GNUe Reports

Introduction

This newsletter mainly covers the the #gnuenterprise IRC channel, with occasional coverage of the three main mailing lists (gnue-announce, gnue and gnue-dev) for the GNU Enterprise project.

1. Further trouble-shooting with the wx 2.6 drivers
20 Jun - 21 Jun Archive Link: "[IRC] 20 Jun 2006"
Summary By Peter Sullivan
Topics: Forms, Common
People: Reinhard Müller, James Thompson, Johannes Vetter, Peter Sullivan

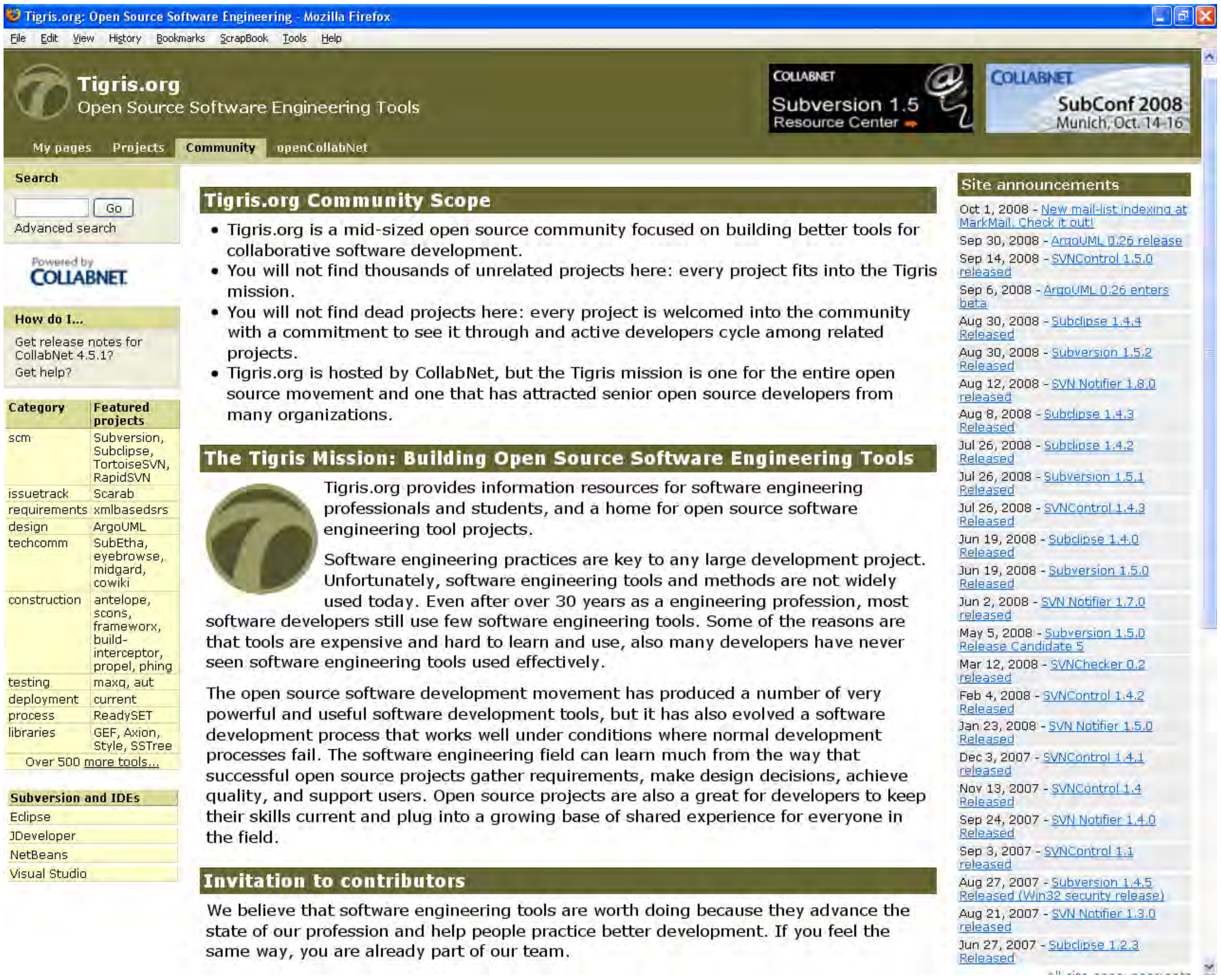
Further to Issue #117, Section #2 (**22 May :** Layout in GNUe Forms with wx 2.6 driver) , Reinhard Müller (reinhard) suggested to James Thompson (jamest) **"if you are bored, you can try again the wx26 uidriver"** , as Johannes Vetter (johannesV) had done **"some massive changes and it might be that your issues with fscking up the boxes are solved"** . James said that, although he was busy, **"i really need to get that tested, as the dropdown box issues in 2.4 are preventing some selections from being allowed"** . So he was keen to have a version of GNUe Forms that worked with the user interface driver for wx 2.6 as soon as possible.

Trying Johannes' new code for GNUe Forms with his existing GNUe Forms Definitions, James found problems - **"none of which are due to anything wrong with what you've done - it's all in my forms"** , where he had been relying on 'features' (such as overlapping text boxes) that Johannes had treated as 'bugs' and now fixed. Johannes confirmed that **"overlaping is now being checked ... not only for boxes but for all widgets"** . He added, **"if you click the detail-button you'll see the offending line in your XML-File - this makes debuging"** a GNUe Form Definition (gfd) **"a lot easier"** . James reported that all five of his existing GNUe Form Definitions were not working with the new code - but **"i would still imagine it's something funky I'm doing in the form"** rather than a problem with Johannes' code. He noted that, on the last one, the problem that he had been having with the dropdown menu had been fixed, but the form now **"aborts on query"** .

(ed. [Peter Sullivan] Note that the lack of any guarantees on backward compatability, even with 'features'/'bugs' is one of the reasons why GNUe Forms remains at a version number below 1.0 as of time of writing, as discussed further in Issue #112, Section #4 (**13 Apr :** Forms approaching version 1.0?) .)

Done

Exhibit 2. A project digest that summarizes multiple messages including those hyperlinked (indicated by highlighted and underlined text fonts) to their originating online sources. (Source: http://www.kerneltraffic.org/GNUe/latest.html, July 2006).



How do I...

Get release notes for CollabNet 4.5.1?

Get help?

Category	Featured projects
scm	Subversion, Subclipse, TortoiseSVN, RapidSVN
issuetrack	Scarab
requirements	xmlbasedsrs
design	ArgoUML
techcomm	SubEtha, eyebrowse, midgard, cowiki
construction	antelope, scons, frameworx, build-interceptor, propel, phing
testing	maxq, aut
deployment	current
process	ReadySET
libraries	GEF, Axion, Style, SSTree
Over 500 more tools...	

Subversion and IDEs

Eclipse

JDeveloper

NetBeans

Visual Studio

Tigris.org Community Scope

- Tigris.org is a mid-sized open source community focused on building better tools for collaborative software development.
- You will not find thousands of unrelated projects here: every project fits into the Tigris mission.
- You will not find dead projects here: every project is welcomed into the community with a commitment to see it through and active developers cycle among related projects.
- Tigris.org is hosted by CollabNet, but the Tigris mission is one for the entire open source movement and one that has attracted senior open source developers from many organizations.

The Tigris Mission: Building Open Source Software Engineering Tools

Tigris.org provides information resources for software engineering professionals and students, and a home for open source software engineering tool projects.

Software engineering practices are key to any large development project. Unfortunately, software engineering tools and methods are not widely used today. Even after over 30 years as a engineering profession, most software developers still use few software engineering tools. Some of the reasons are that tools are expensive and hard to learn and use, also many developers have never seen software engineering tools used effectively.

The open source software development movement has produced a number of very powerful and useful software development tools, but it has also evolved a software development process that works well under conditions where normal development processes fail. The software engineering field can learn much from the way that successful open source projects gather requirements, make design decisions, achieve quality, and support users. Open source projects are also a great for developers to keep their skills current and plug into a growing base of shared experience for everyone in the field.

Invitation to contributors

We believe that software engineering tools are worth doing because they advance the state of our profession and help people practice better development. If you feel the same way, you are already part of our team.

Site announcements

Oct 1, 2008 - New mail-list indexing at MarkMail. Check it out!
Sep 30, 2008 - ArgoUML 0.26 release
Sep 14, 2008 - SVNControl 1.5.0 released
Sep 6, 2008 - ArgoUML 0.26 enters beta
Aug 30, 2008 - Subclipse 1.4.4 Released
Aug 30, 2008 - Subversion 1.5.2 Released
Aug 12, 2008 - SVN Notifier 1.8.0 released
Aug 8, 2008 - Subclipse 1.4.3 Released
Jul 26, 2008 - Subclipse 1.4.2 Released
Jul 26, 2008 - Subversion 1.5.1 Released
Jul 26, 2008 - SVNControl 1.4.3 Released
Jun 19, 2008 - Subclipse 1.4.0 Released
Jun 19, 2008 - Subversion 1.5.0 Released
Jun 2, 2008 - SVN Notifier 1.7.0 released
May 5, 2008 - Subversion 1.5.0 Release Candidate 5
Mar 12, 2008 - SVNChecker 0.2 released
Feb 4, 2008 - SVNControl 1.4.2 Released
Jan 23, 2008 - SVN Notifier 1.5.0 Released
Dec 3, 2007 - SVNControl 1.4.1 released
Nov 13, 2007 - SVNControl 1.4 Released
Sep 24, 2007 - SVN Notifier 1.4.0 Released
Sep 3, 2007 - SVNControl 1.1 released
Aug 27, 2007 - Subversion 1.4.5 Released (Win32 security release)
Aug 21, 2007 - SVN Notifier 1.3.0 released
Jun 27, 2007 - Subclipse 1.2.3 Released

Exhibit 3. A view of the mission of the Tigris.org project community and its intent to span and bring together the FOSSD and SE communities (Source: http://www.tigris.org, accessed October 2008).